



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Curso: Principios de Sistemas Operativos

II Semestre 2023

Proyecto II

Estudiantes:

Celina Madrigal Murillo - 2020059364

María José Porras Maroto - 2019066056

Profesor:

Armando Arce Orozco

Grupo: 2

Fecha de entrega:

Jueves 26 de Octubre del 2023

Introducción

En el presente documento se expone la documentación correspondiente al proyecto 2 del curso Principios de Sistemas Operativos. Este proyecto tiene como objetivo la creación de un programa denominado "star" (Simple Tar), el cual emula la funcionalidad del comando tar en entornos UNIX. Este programa permitirá la consolidación de múltiples archivos en un solo archivo, facilitando su almacenamiento y manipulación.

La implementación interna del archivo se basa en "memoria contigua", organizando los archivos consecutivamente. Se lleva un control preciso de la ubicación de inicio y fin de cada archivo, permitiendo una gestión eficiente de espacios libres. Asimismo, se brinda la opción de actualizar el contenido de un archivo existente. La herramienta 'star' se distingue por su capacidad de desfragmentación, liberando espacios no utilizados sin necesidad de archivos temporales.

Descripción del problema

El objetivo de este proyecto consiste en programar un emparador de archivos. Este es el tipo de funcionalidad que provee el comando tar en ambientes UNIX.

El programa tar, es usado para almacenar múltiples archivos en un solo archivo. Dentro de los entornos Unix tar aparece como un comando que puede ser ejecutado desde la línea de comandos de una consola de texto o desde un simple terminal. El formato del comando tar es, comúnmente

```
tar <opciones> <archivoSalida> <archivo1> <archivo2> ... <archivoN>
```

donde <archivoSalida> es el archivo resultado y <archivo1>, <archivo2>, etc; son los diferentes archivos que serán "empaquetados" en <archivoSalida>.

Las opciones más comunes son las siguientes:

- -c, --create : crea un nuevo archivo
- -x, --extract : extraer de un archivo
- -t, --list: listar los contenidos de un archivo

- `--delete`: borrar desde un archivo
- `-u, --update`: actualiza el contenido del archivo
- `-v, --verbose`: ver un reporte de las acciones a medida que se van realizando
- `-f, --file`: empacar contenidos de archivo, si no está presente asume la entrada estándar.
- `-r, --append`: agrega contenido a un archivo
- `-p, --pack`: desfragmenta el contenido del archivo (no presente en `tar`)

Ejemplos

1. Si queremos empacar un archivo llamado "index.html" y guardar los datos en "html-paq.tar", lo haríamos con la instrucción

```
tar -cvf html-paq.tar index.html
```
2. Si queremos desempaquetar todo el contenido de un archivo llamado xxx.tar podemos utilizar un comando como este

```
tar -xvf xxx.tar
```
3. Para archivar el contenido de tres archivos doc1.txt, doc2.txt y data.dat

```
tar -cvf foo.tar doc1.txt doc2.txt data.dat
```
4. Si ahora se desea eliminar el contenido del archivo data.dat se ejecutaría

```
tar --delete -vf foo.tar data.dat
```
5. Para agregar ahora un nuevo archivo test.doc a foo.tar se ejecutaría

```
tar -rvf foo.tar test.doc
```
6. Los contenidos se desempaquetarán en el directorio actual.

Programación

Se deberá programar el comando `star` ("simple tar", NO "estrella"), de tal forma que acepte los comandos básicos mostrados anteriormente. Para desarrollar su programa usted debe tomar en cuenta los siguientes aspectos:

- El archivo debe organizarse internamente utilizando "memoria contigua". Esto significa que los archivos a agregar se almacenan (al principio) seguidos en el archivo empacado. Note que debe llevar control del byte en donde empieza cada archivo y el byte en donde termina.

- Al crear un archivo empacado este se crea del tamaño necesario para almacenar los archivos agregados. Cuando se borra algún contenido, el archivo empacado no cambia de tamaño sino que se lleva registro de los espacios liberados. Si posteriormente se agrega un nuevo contenido entonces se reutiliza el espacio libre. Si aún así el nuevo contenido no cabe, se hace crecer el archivo empacado. No debe utilizar ningún archivo auxiliar para hacer crecer el archivo.
- Tome en cuenta que un archivo que se agrega puede ya existir en el archivo empacado. Es decir, lo que se desea hacer es actualizar su contenido. Para esto existe la opción `update (-u)` que sobrescribirá el contenido de un archivo.
- Se debe llevar un control de los espacios libres, para asignar los huecos libres se utilizará la técnica del primer ajuste o el siguiente ajuste. Note que los espacios libres contiguos se deben fusionar.
- Este programa no utilizará los derechos de acceso, que normalmente almacenaría un archivo empacado tar en ambiente UNIX.
- La opción de desfragmentación `(-p)` no es estándar (no está presente en tar) y lo que hace es desfragmentar el contenido almacenado en el archivo empacado y liberar cualquier espacio sin utilizar. Es decir, con este comando se liberarán todos los bloques libres y el tamaño del archivo empacado se ajustará al contenido real existente. Note que no se debe utilizar un archivo temporal para realizar esta función, toda la desfragmentación se debe realizar sobre el contenido del mismo archivo.
- La opción `-v` muestra información sobre la operación ejecutada. Se puede aplicar dos veces `-vv` para ver información adicional.
`tar -rvf foo.tar test.doc`

Definición de estructuras de datos

El proyecto hace uso de dos estructuras fundamentales para llevar a cabo sus operaciones: ``File`` y ``PackageInfo``.

Estructura `File`

La estructura `File` representa un archivo que será empacado por el programa. Contiene los siguientes campos:

- `name[MAX_FILENAME_LENGTH]`: Una cadena de caracteres que almacena el nombre del archivo. `MAX_FILENAME_LENGTH` define la longitud máxima permitida para el nombre del archivo.
- `size`: Un entero sin signo (`size_t`) que indica el tamaño del archivo en bytes.
- `start`: Un valor de tipo `long` que indica el byte de inicio del archivo dentro del archivo empacado.
- `end`: Un valor de tipo `long` que indica el byte de fin del archivo dentro del archivo empacado.

Estructura `PackageInfo`

La estructura `PackageInfo` contiene información acerca de los archivos empacados. Incluye:

- `files[MAX_FILE_COUNT]`: Un arreglo de estructuras `File` que almacena los archivos empacados. `MAX_FILE_COUNT` determina la cantidad máxima de archivos que se pueden empaquetar en un único archivo.
- `file_count`: Un entero sin signo (`size_t`) que representa el número de archivos empacados en la estructura.

Estas estructuras proporcionan la base para la gestión de archivos y la organización de la información dentro del archivo empacado. Permiten llevar un registro detallado de cada archivo, incluyendo su nombre, tamaño y ubicación dentro del archivo empacado. Esto facilita la manipulación eficiente de los archivos y la implementación de las funcionalidades requeridas por el proyecto.

```
//Estructuras
struct File {
    char name[MAX_FILENAME_LENGTH];
    size_t size;
    long start;
    long end;
};

struct PackageInfo {
    struct File files[MAX_FILE_COUNT];
    size_t file_count;
};
```

Descripción detallada y explicación de los componentes principales del programa

Mecanismo de acceso a archivos

El programa implementa un conjunto de funciones que manipulan archivos de acuerdo con las necesidades del proyecto. Esto incluye la creación, lectura, escritura, extracción, eliminación y actualización de archivos dentro de un archivo empaado. Cada una de estas operaciones se apoya en las funciones y mecanismos proporcionados por las bibliotecas estándar de C para el manejo de archivos. Algunas de estas son las siguientes:

- El programa abre archivos utilizando la función `fopen()`. Por ejemplo, en la función `crear()` se utiliza `FILE* package = fopen(package_name, "wb");` para abrir un archivo en modo escritura binaria ("wb"). Esto crea un puntero a un objeto de tipo `FILE` que representa el archivo abierto.
- Se utiliza la función `fread()` para leer datos desde un archivo. Por ejemplo, `size_t leidos = fread(empacado, sizeof(struct PackageInfo), 1, archivo);` se utiliza para leer la estructura `PackageInfo` desde el archivo empaado.

- Para escribir en un archivo, se emplea la función `fwrite()`. Por ejemplo, `fwrite(&empacado, sizeof(struct PackageInfo), 1, archivo)`; se usa para escribir la estructura `PackageInfo` en el archivo.
- `fseek()` permite mover el indicador de posición del archivo. Por ejemplo, `fseek(file, 0, SEEK_END)`; se utiliza para mover el puntero al final del archivo.
- `rewind()` vuelve al inicio del archivo. Por ejemplo, `rewind(file)`; se emplea para volver al inicio del archivo.
- Para liberar los recursos asociados con un archivo, se utiliza `fclose()`. Por ejemplo, `fclose(file)`; cierra el archivo `file`.
- `ftruncate()` se emplea para truncar un archivo a un tamaño específico. Por ejemplo, `ftruncate(fileno(truncated_file), new_size)`; se utiliza para truncar el archivo a un nuevo tamaño.

Estructura de directorios internos

Las dos estructuras 'File' y 'PackageInfo' desempeñan un papel importante en la forma de organización del proyecto, especialmente, en el manejo dentro del archivo empacado. En conjunto, cumplen la función de proporcionar una organización lógica de los archivos empacados que se van generando. En este sentido, y según lo descrito en las secciones anteriores, la estructura 'File' representa y contiene la información de los archivos individuales que se agruparán dentro del archivo contenedor. Mientras que por su parte, 'Packageinfo' es quien actúa directamente como "directorio" de los archivos empaquetados y quien va manejando los 'File' asociados, así como quien nos permite determinar el resto de información necesaria para la realización del resto de comandos.

De esta forma 'File' y 'Packageinfo' estas estructuras permiten organizar y gestionar eficientemente los archivos empaquetados dentro del archivo contenedor. En resumen, estas estructuras funcionan como una estructura de organización que permite llevar un registro de los archivos empaquetados, sus nombres y ubicaciones, lo que facilita la gestión y manipulación de los archivos dentro del archivo empaquetado.

Estrategia de administración de espacios libres

La estrategia de administración de espacios libres se basa en encontrar y asignar bloques de espacio libre dentro del archivo empacado para almacenar archivos. La administración se realiza en la función ``encontrar_espacios_libres``.

"Inicio" se inicializa con el tamaño de la estructura ``PackageInfo``, que es el primer espacio disponible después de la información de metadatos. Se itera sobre todos los archivos existentes en el paquete. Se calcula el ``fin``, que es el inicio del próximo archivo o, si es el último, el final del archivo empacado. Se verifica si hay suficiente espacio entre ``inicio`` y ``fin`` para acomodar el nuevo archivo (``fin - inicio >= archivo_tamano``). Si hay suficiente espacio, se asigna este bloque de espacio libre al nuevo archivo. Si no hay suficiente espacio, se continúa con el próximo archivo. Si no se encontró un espacio libre lo suficientemente grande en los archivos existentes, se crea uno al final del archivo empacado, siempre y cuando el tamaño del archivo no exceda ``MAX_FILE_SIZE``. Si no hay suficiente espacio en los archivos existentes, se agrega un nuevo bloque al final del archivo.


```

void encontrar_espacios_libres(struct PackageInfo *empacado, long archivo_tamano) {

    verbose("Inicia la búsqueda de espacios libres",1);

    //Inicializa el inicio del espacio libre
    long inicio = sizeof(struct PackageInfo);

    //Recorre los archivos del paquete
    for (size_t i = 0; i < empacado->file_count; i++) {

        //Calcula el final del espacio libre actual
        long fin = empacado->files[i].start;
        if (i + 1 < empacado->file_count) {
            fin = empacado->files[i + 1].start;
        }

        //Si el espacio libre actual es lo suficientemente grande, lo asigna
        if (fin - inicio >= archivo_tamano) {

            verbose("Encuentra un espacio libre de tamaño suficiente",1);

            //Asigna el espacio libre
            empacado->files[empacado->file_count].start = inicio;
            empacado->files[empacado->file_count].end = inicio + archivo_tamano;
            empacado->file_count++;

            //Termina la función
            return;
        }

        //Actualiza el inicio del espacio libre
        inicio = empacado->files[i].end;
    }

    //Si no se encontró un espacio libre lo suficientemente grande, lo crea al final
    if (inicio + archivo_tamano <= MAX_FILE_SIZE) {

        //Verbose
        verbose("No se encontró un espacio libre lo suficientemente grande, crea uno al final",1);
        verbose("Crea un espacio libre", 2);

        //Crea el espacio libre
        empacado->files[empacado->file_count].start = inicio;
        empacado->files[empacado->file_count].end = inicio + archivo_tamano;
        empacado->file_count++;
    }
}

```

Procedimiento de desfragmentación del archivo

La desfragmentación aquí se refiere a la reorganización de los archivos dentro del archivo empacado para que estén ubicados contiguamente en el espacio de almacenamiento. Antes de llevar a cabo la desfragmentación, los archivos se ordenan por la posición de inicio. Esto significa que los archivos estarán ubicados en el archivo empacado en orden ascendente según sus posiciones de inicio. Se itera

sobre los archivos y se verifica si la posición de inicio actual es diferente de la posición que debería tener según el orden. Si son diferentes, significa que el archivo no está en la posición correcta y se necesita mover.

- Se mueve el puntero del archivo empacado a la posición de inicio del archivo que se va a mover.
- Se lee el contenido del archivo en un buffer.
- Se mueve el puntero del archivo empacado a la nueva posición.
- Se escribe el contenido del buffer en la nueva posición.

Esto efectivamente mueve el archivo dentro del archivo empacado a su nueva ubicación.

Una vez que se han movido todos los archivos según sea necesario, se actualiza la información de metadatos (la estructura `PackageInfo`) con las nuevas posiciones de inicio y fin de cada archivo.

```
void desfragmentar(const char* package_name) {  
  
    verbose("Inicia opción de desfragmentar",1);  
    verbose("Abre el archivo empacado en modo lectura/escritura binaria",2);  
  
    FILE* package = fopen(package_name, "rb+");  
    if (package == NULL) {  
        perror("No se pudo abrir el archivo empacado");  
        exit(1);  
    }  
  
    verbose("Lee la estructura",1);  
  
    struct PackageInfo empacado;  
    size_t leidos = fread(&empacado, sizeof(struct PackageInfo), 1, package);  
  
    if (leidos != 1) {  
        perror("Error al leer el archivo");  
        fclose(package);  
        return;  
    }  
  
    verbose("Ordena los archivos por inicio",2);  
  
    for (size_t i = 0; i < empacado.file_count - 1; i++) {  
        for (size_t j = i + 1; j < empacado.file_count; j++) {  
            if (empacado.files[j].start < empacado.files[i].start) {  
                struct File temp = empacado.files[i];  
                empacado.files[i] = empacado.files[j];  
                empacado.files[j] = temp;  
            }  
        }  
    }  
}
```

```

verbose("Mueve los archivos si es necesario",2);

long current_byte = sizeof(struct PackageInfo);
for (size_t i = 0; i < empacado.file_count; i++) {
    if (current_byte != empacado.files[i].start) {

        verbose("Mueve el archivo",1);

        fseek(package, empacado.files[i].start, SEEK_SET);
        char buffer[MAX_FILE_SIZE];
        size_t bytes_read = fread(buffer, 1, empacado.files[i].size, package);
        fseek(package, current_byte, SEEK_SET);
        fwrite(buffer, 1, bytes_read, package);

        empacado.files[i].start = current_byte;
        empacado.files[i].end = current_byte + empacado.files[i].size;
    }

    current_byte = empacado.files[i].end;
}

verbose("Actualiza la información de los archivos en el paquete",1);

fseek(package, 0, SEEK_SET);
fwrite(&empacado, sizeof(struct PackageInfo), 1, package);

printf("Paquete desfragmentado\n");

verbose("Cierra el archivo empacado",1);

fclose(package);
}

```

Análisis de resultados de pruebas (funcionamiento)

En esta sección se presentan algunas de las pruebas realizadas y los resultados obtenidos en cada una. Estas pruebas fueron divididas por cada comando, con el objetivo de visualizar el funcionamiento de cada opción dentro del proyecto.

Crear

Para corroborar el comportamiento del comando crear se busca generar un archivo "EjemploSalida.star" que contenga los archivos "Paises.txt" y "Ciudades.txt". El comando que se escribe en consola es "-cvf", donde "v" nos va a dar reportes

sobre el funcionamiento de la operación, “c” es quien inicia la función de crear y “f” nos permite indicar los archivos a adjuntar.

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ gcc -o star star.c
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -cvf EjemploSalida.star Países.txt Ciudades.txt
Inicia opción de crear
Comienza ciclo de archivos
Obtiene tamaño del archivo
Agrega archivo
Obtiene tamaño del archivo
Agrega archivo
Cierra el archivo empaçado
Paquete Creado con éxito.
```

Si se revisa la carpeta donde se encuentra el programa star.c encontraremos el archivo nuevo “EjemploSalida.star”

EjemploSalida.star	25/10/2023 01:41 a. m.	Archivo STAR	28 KB
--------------------	------------------------	--------------	-------

Agregar

Al paquete “EjemploSalida.star” se le desea agregar también el archivo “Productos.txt”, por lo que se utilizara la opción de append creada. Para verificar su correcta incorporación se ingresará el comando “-tf” seguido del comando “-rft”, con el objetivo de verificar los archivos anteriores y posteriores a la inserción de “Productos.txt”.

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -tf EjemploSalida.star
Nombre de archivo: Países.txt
Nombre de archivo: Ciudades.txt
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -rft EjemploSalida.star Productos.txt
Archivo agregado con éxito.
Nombre de archivo: Países.txt
Nombre de archivo: Ciudades.txt
Nombre de archivo: Productos.txt
```

Luego de la ejecución tenemos como resultado el siguiente archivo empaçado. Este archivo al no haber realizado ningún tipo de borrado no posea espacios libres, por lo que se tuvo que insertar el archivo “Productos.txt” al final del documento. Lo que implica un aumento en el tamaño inicial del paquete.

EjemploSalida.star	25/10/2023 01:49 a. m.	Archivo STAR	30 KB
--------------------	------------------------	--------------	-------

Igualmente, podemos observar que el tamaño del archivo cambió en consecuencia con el contenido agregado. En este caso de prueba el archivo “Productos.txt” tenía un tamaño de 2k, por lo que el paquete aumentó de 28k a 30k.

Productos	23/10/2023 02:30 p. m.	Documento de texto	2 KB
-----------	------------------------	--------------------	------

Listar

Con este comando se pueden listar los archivos contenidos dentro del paquete generado. Para corroborar su funcionamiento lo que debe de hacer es ingresar el comando “-tf” seguido del nombre del paquete del que se desea conocer el contenido.

A continuación se agregan tres ejemplos de otros paquetes y las diferencias en contenido que cada uno posee y como este las muestra.

Creación de paquetes

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -cvf Empaque.star Menu.txt
Inicia opcion de crear
Comienza ciclo de archivos
Obtiene tamaño del archivo
Agrega archivo
Cierra el archivo empacado
Paquete Creado con éxito.
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -cvf Prueba.star Paises.txt Ciudades.txt
Inicia opcion de crear
Comienza ciclo de archivos
Obtiene tamaño del archivo
Agrega archivo
Obtiene tamaño del archivo
Agrega archivo
Cierra el archivo empacado
Paquete Creado con éxito.
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -cvf Paquete.star Menu.txt Restaurantes.txt
Inicia opcion de crear
Comienza ciclo de archivos
Obtiene tamaño del archivo
Agrega archivo
Obtiene tamaño del archivo
Agrega archivo
Cierra el archivo empacado
Paquete Creado con éxito.
```

Listado de contenidos

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -tf Empaque.star
Nombre de archivo: Menu.txt
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -tf Paquete.star
Nombre de archivo: Menu.txt
Nombre de archivo: Restaurantes.txt
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -tf Prueba.star
Nombre de archivo: Paises.txt
Nombre de archivo: Ciudades.txt
```

Verbose

Con el objetivo de visualizar la realización de cada uno de los procedimientos es que se creó la función de verbose. El comando -v muestra información sobre la operación ejecutada. Si la letra v es ingresada dos veces este genera reportes de información adicional a notar en la ejecución del procedimiento.

Verbose simple

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -cvf Nuevo.star Países.txt Ciudades.txt
Inicia opción de crear
Comienza ciclo de archivos
Obtiene tamaño del archivo
Agrega archivo
Obtiene tamaño del archivo
Agrega archivo
Cierra el archivo empaçado
Paquete Creado con éxito.
```

Verbose ampliado

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -cvvf Nuevo2.star Países.txt Ciudades.txt
Inicia opción de crear
Obtiene información del archivo
Comienza ciclo de archivos
Obtiene tamaño del archivo
Reinicia puntero del archivo
Copia nombre del archivo
Agrega archivo
Copia contenido del archivo al archivo empaçado
Obtiene tamaño del archivo
Reinicia puntero del archivo
Copia nombre del archivo
Agrega archivo
Copia contenido del archivo al archivo empaçado
Cierra el archivo empaçado
Paquete Creado con éxito.
```





Con los dos ejemplos anteriores se puede visualizar mejor la diferencia en consola sobre los dos tipos de información que se pueden tener sobre un mismo comando.

Extraer

Al extraer los archivos de nuestro empaçado lo que se debe de hacer es escribir el comando “-xf” y especificar el nombre del paquete del que se desea extraer el contenido.

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -xf EjemploSalida.star
Archivos extraídos con éxito.
```

Lo que resulta en la escritura de contenido en los archivos presentes en el empaquetado y mantiene el paquete de la misma manera en la que se encontraba con anterioridad, es decir, sobre el paquete no se genera ningún cambio.

	Ciudades	25/10/2023 01:59 a. m.	Documento de texto	1 KB
	Países	25/10/2023 01:59 a. m.	Documento de texto	1 KB
	Productos	25/10/2023 01:59 a. m.	Documento de texto	1 KB
	EjemploSalida.star	25/10/2023 01:49 a. m.	Archivo STAR	30 KB

Borrar

Para ejemplificar el uso del comando de borrado, se eliminará el archivo “Productos.txt” del paquete generado. Con el uso del comando “-dft” se eliminará el

contenido del archivo de texto y también veremos cuáles archivos siguen presentes en el empaçado.

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -tf EjemploSalida.star
Nombre de archivo: Países.txt
Nombre de archivo: Ciudades.txt
Nombre de archivo: Productos.txt
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -dft EjemploSalida.star Productos.txt
El archivo "Productos.txt" ha sido eliminado del paquete.
Nombre de archivo: Países.txt
Nombre de archivo: Ciudades.txt
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$
```

En este caso hubo un cambio en el tamaño del archivo, pero este se da por el cambio del PackageInfo que se tenía con anterioridad.

 EjemploSalida.star	25/10/2023 02:02 a. m.	Archivo STAR	29 KB
--	------------------------	--------------	-------

Actualizar

Actualizar en el proyecto se tiene acceso mediante el comando “-uf” y el nombre del paquete y archivo que se desea modificar.

```
→ Pr2-S0 git:(master) x ./star -uf EjemploSalida.star Países.txt
El archivo "Países.txt" ha sido actualizado en el paquete.
```

El mensaje nos indica que el archivo ha sido actualizado con éxito.

Desfragmentar

Al desfragmentar nuestro empaçado lo que se debe de hacer es escribir el comando “-pf” y especificar el nombre del paquete del que se desea desfragmentar el contenido.

```
majo@LAPTOP-HM49NOP7:/mnt/c/Principios/Proyecto2/Pr2-S0$ ./star -pfvv EjemploSalida.star
Inicia opción de desfragmentar
Abre el archivo empaçado en modo lectura/escritura binaria
Lee la estructura
Ordena los archivos por inicio
Mueve los archivos si es necesario
Actualiza la información de los archivos en el paquete
Paquete desfragmentado
Cierra el archivo empaçado
```

Con la opción de verbose podemos ver un desglose más detallado de las acciones que se llevaron a cabo en la función desde abrir el archivo hasta lograr su desfragmentación.

Conclusiones

A través de la elaboración del proyecto se ha generado un entendimiento más profundo sobre cómo se pueden organizar los archivos en términos de sistemas de archivos y como el tipo de organización nos permite combinar varios en uno solo. Esto implica comprender la estructura de almacenamiento de archivos en memoria contigua y cómo se pueden manejar diferentes tipos de archivos y metadatos relacionados. Igualmente, aunque no se trata de directorios tradicionales, la organización de archivos en estructuras como 'PackageInfo' y 'File' proporciona una comprensión de cómo se pueden crear jerarquías lógicas para los archivos, lo que es fundamental en la gestión de sistemas de archivos.

El desarrollo de este proyecto implicaba también el entendimiento y aplicación de temas como el manejo de archivos y la gestión de espacios libres dentro del empaquetado. Las operaciones de lectura y escritura implican la manipulación de datos dentro de un archivo empaquetado, lo que a su vez afecta el espacio en disco utilizado. Al entender cómo funcionan estas operaciones y cómo se administran los recursos en memoria contigua, nos permitió que se pudieran tomar decisiones informadas sobre cómo asignar y liberar espacio en disco de manera eficiente, lo que es esencial para garantizar un uso óptimo de los recursos de almacenamiento.

Por último, la implementación de cada una de las funcionalidades y de los comandos generados resultaron en un gran ejercicio en términos de decisiones de diseño. La escogencia de estructuras, la organización de los archivos, los algoritmos y estrategias adoptados en cada uno de los comandos y funciones auxiliares implican el entendimiento y atención de los participantes. Esto no solo aportó a mejorar el aprendizaje sino que permitió poner en práctica los conceptos y temas enseñados.